OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

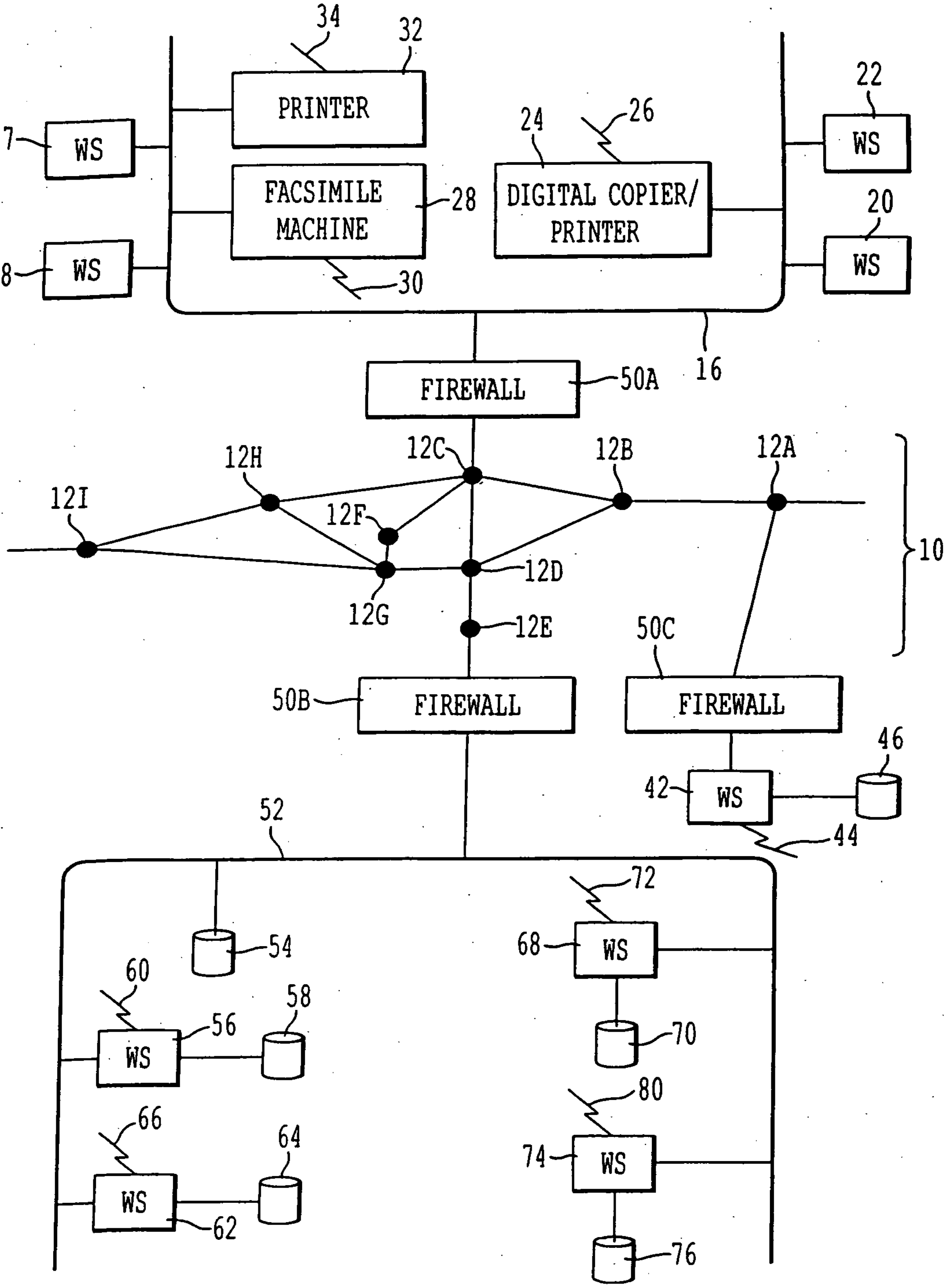


FIG. 1
BACKGROUND ART

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

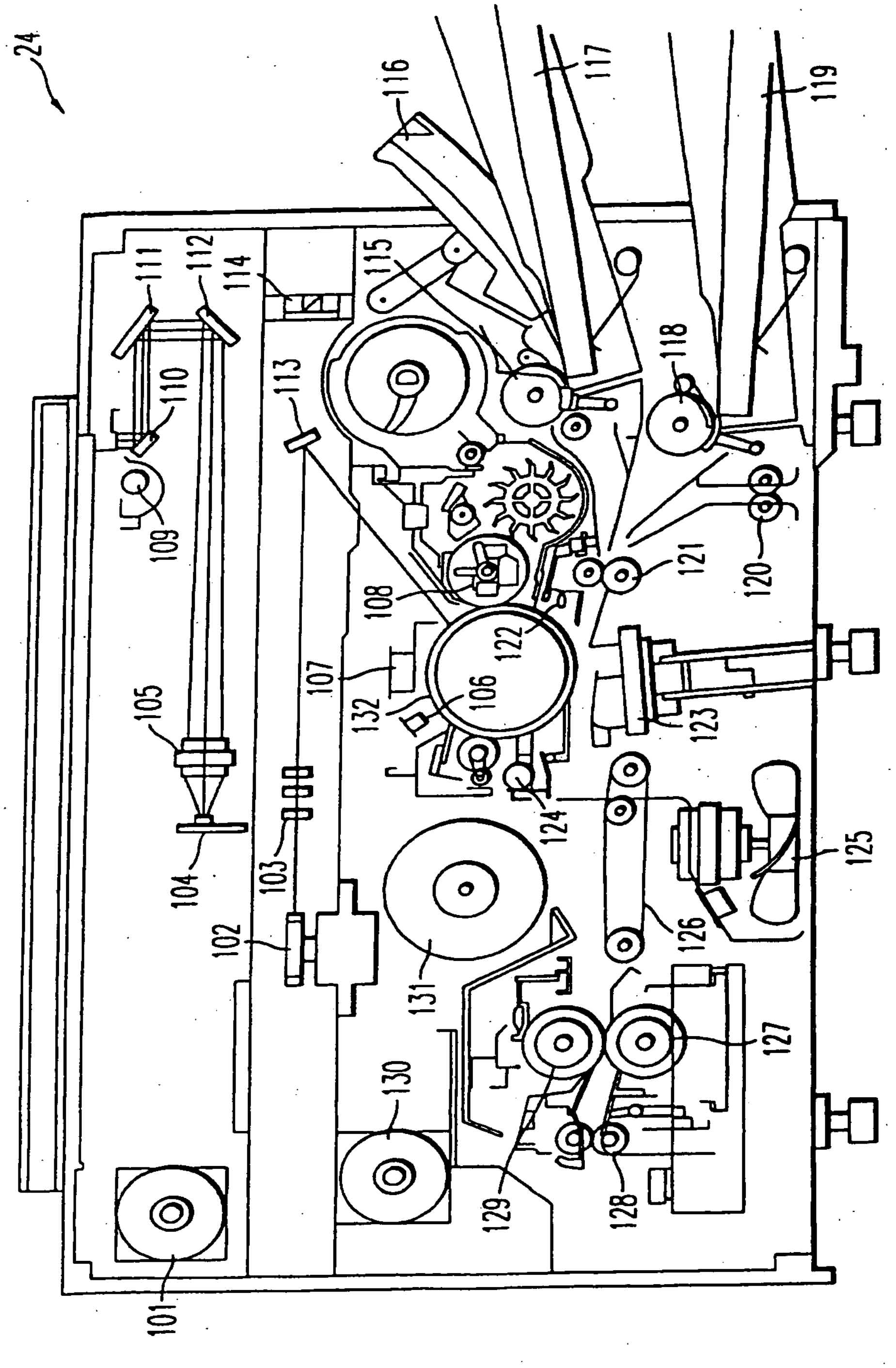


FIG. 2
BACKGROUND ART

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

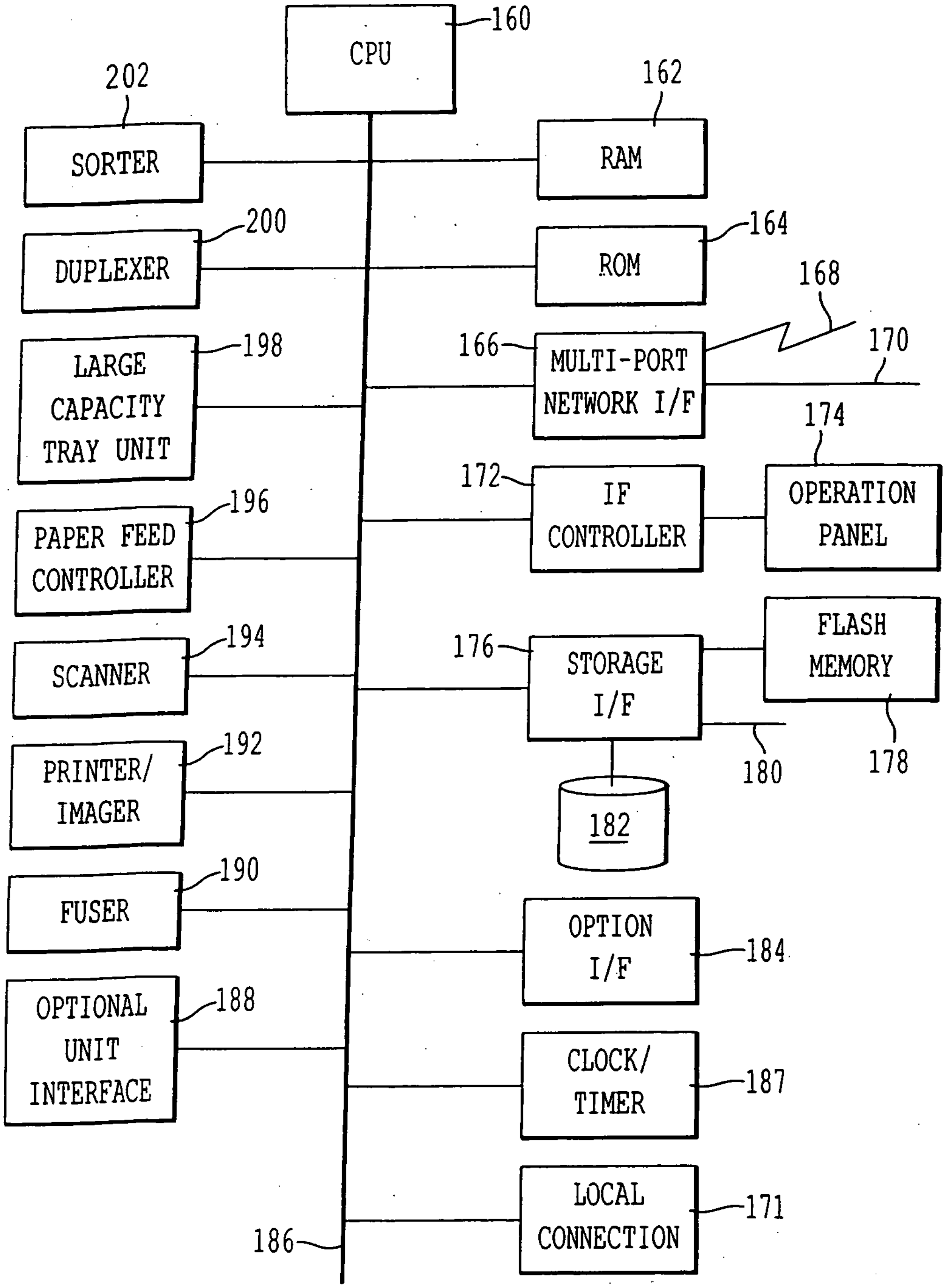


FIG. 3

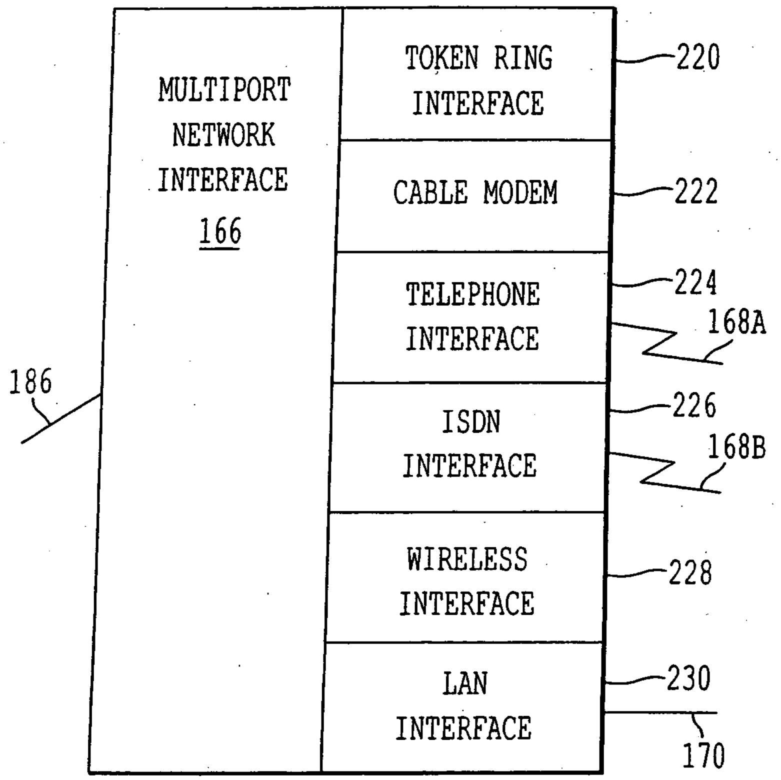

MULTIPORT NETWORK INTERFACE
166
TOKEN RING INTERFACE
220
CABLE MODEM
222
TELEPHONE INTERFACE
224
168A
ISDN INTERFACE
226
168B
WIRELESS INTERFACE
228
LAN INTERFACE
230
170
186

FIG. 4

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

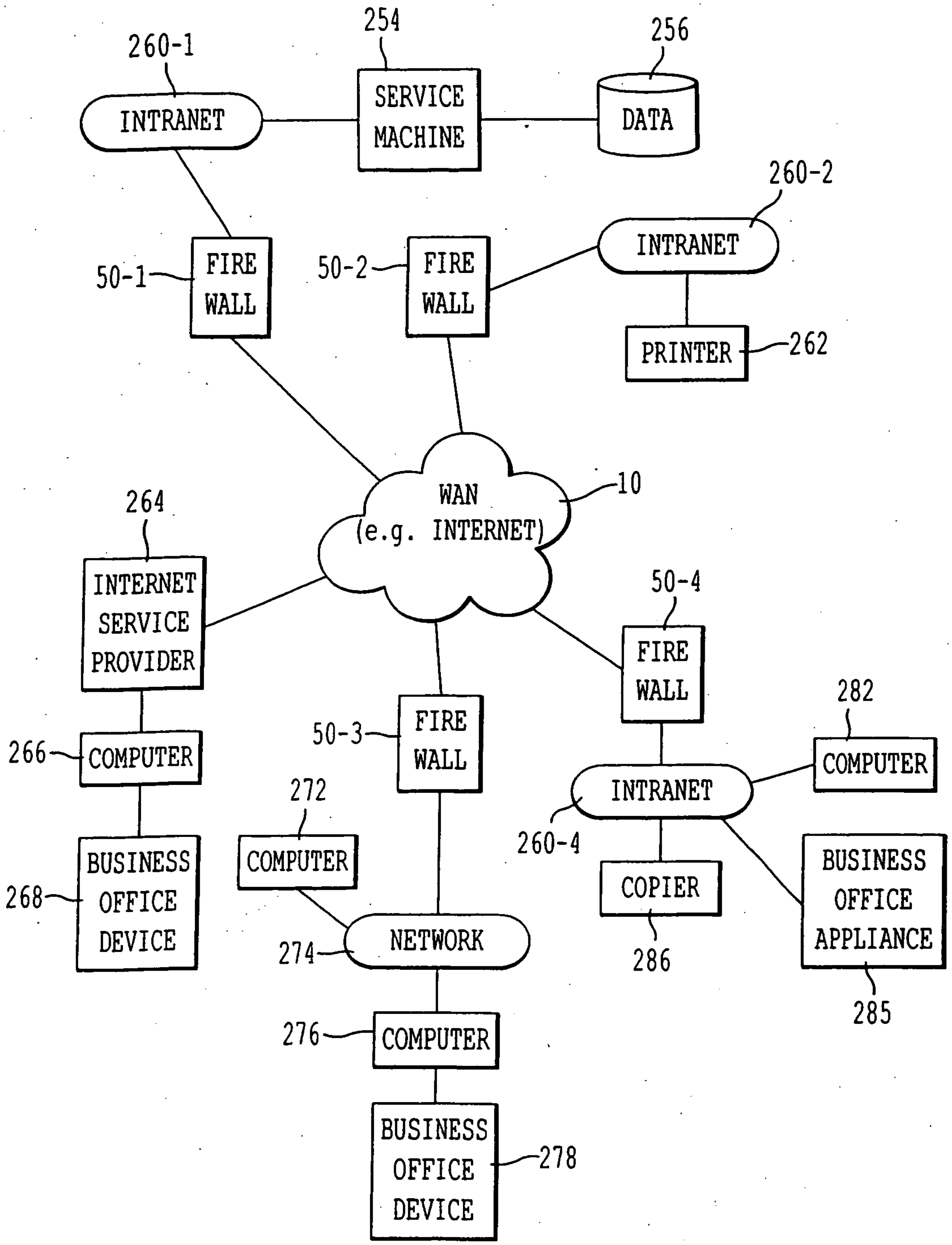


FIG. 5

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

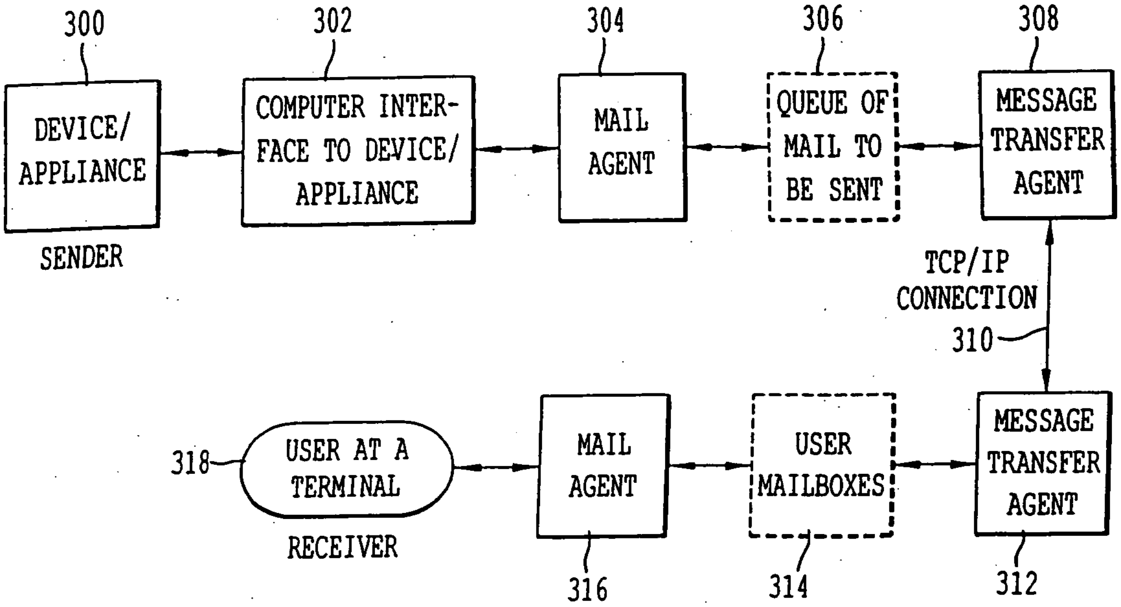


FIG. 6A

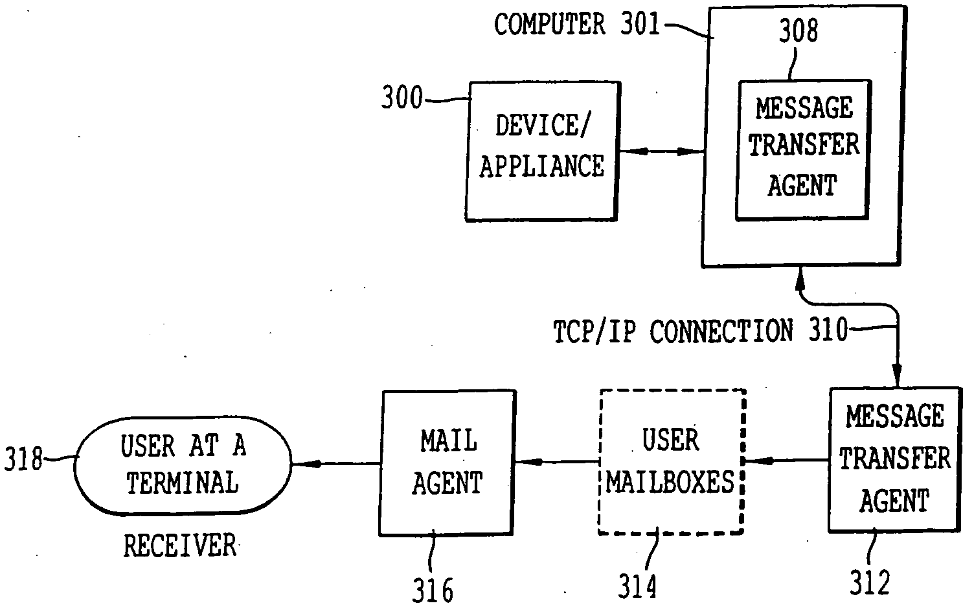


FIG. 6B

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

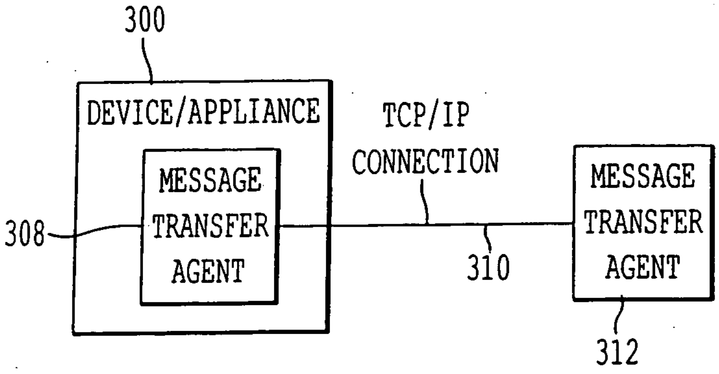


FIG. 6C

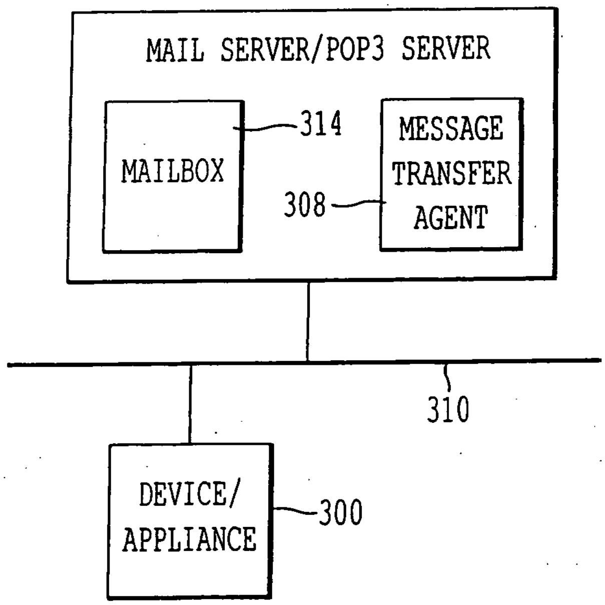


FIG. 6D

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

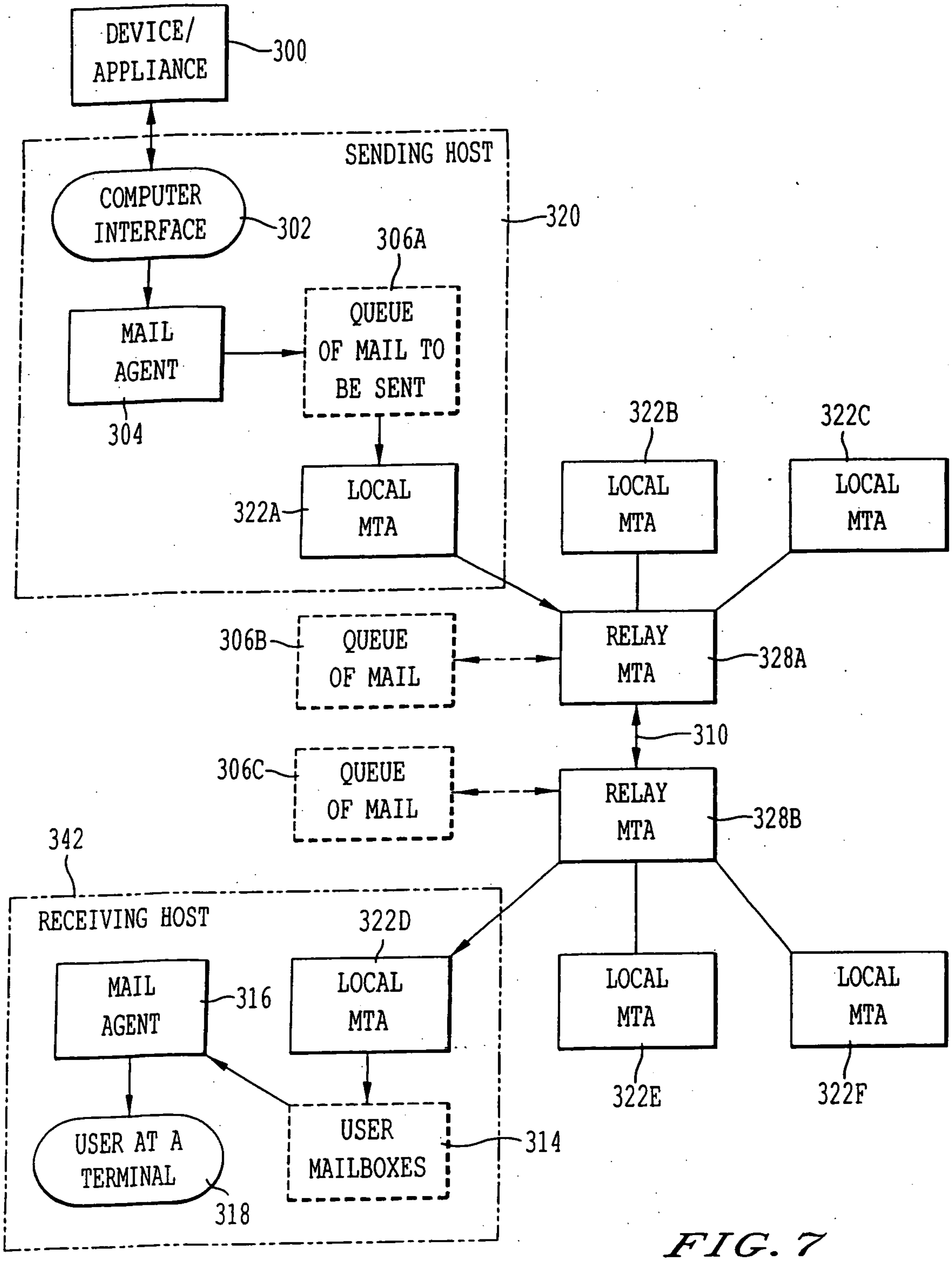


FIG. 7

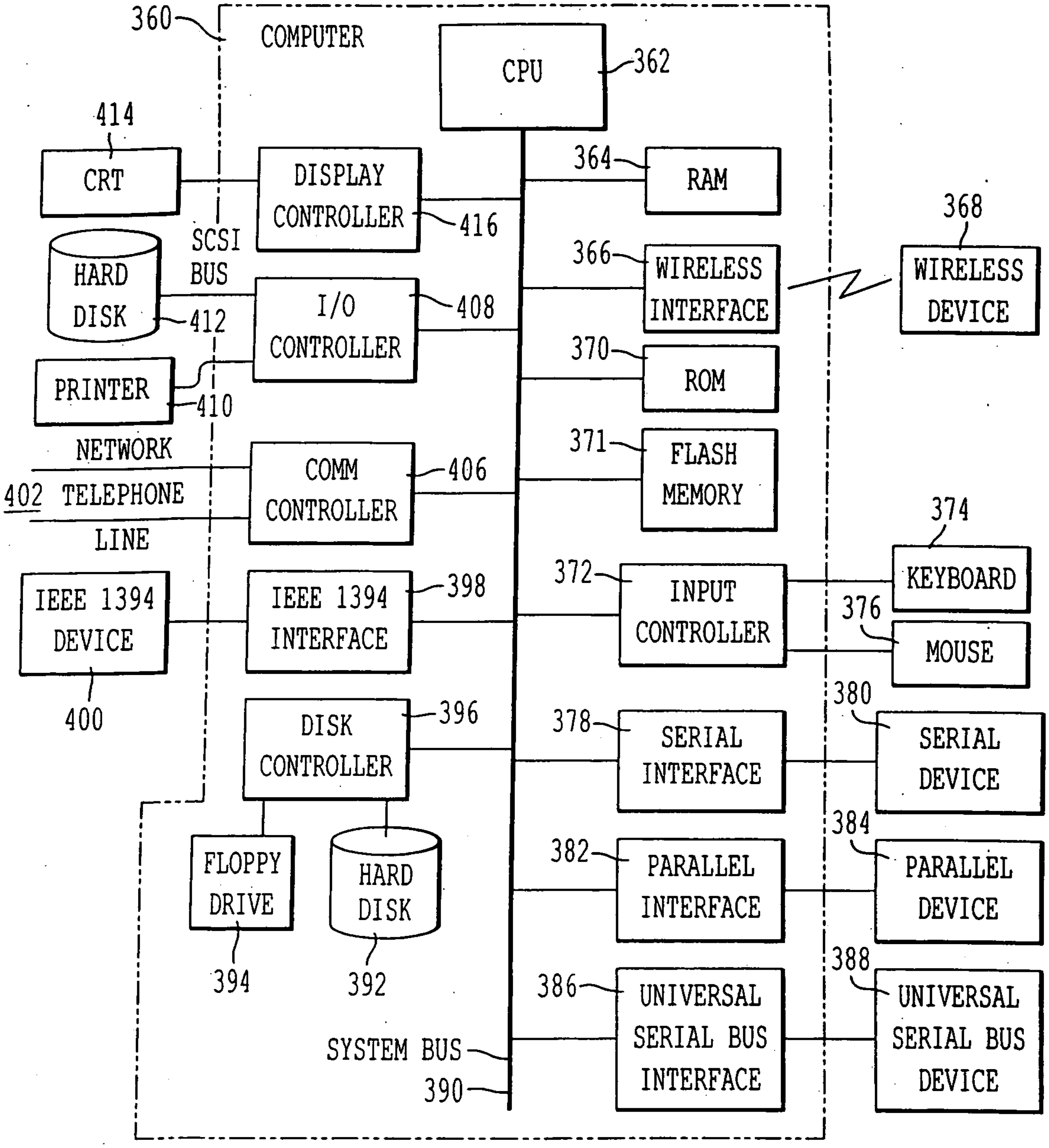

360
COMPUTER
CPU
362
414
CRT
DISPLAY CONTROLLER
416
364
RAM
368
WIRELESS DEVICE
SCSI BUS
HARD DISK
412
I/O CONTROLLER
408
366
WIRELESS INTERFACE
370
ROM
PRINTER
410
NETWORK
402
TELEPHONE LINE
COMM CONTROLLER
406
371
FLASH MEMORY
374
KEYBOARD
372
INPUT CONTROLLER
376
MOUSE
IEEE 1394 DEVICE
400
IEEE 1394 INTERFACE
398
DISK CONTROLLER
396
378
SERIAL INTERFACE
380
SERIAL DEVICE
382
PARALLEL INTERFACE
384
PARALLEL DEVICE
FLOPPY DRIVE
394
HARD DISK
392
386
UNIVERSAL SERIAL BUS INTERFACE
388
UNIVERSAL SERIAL BUS DEVICE
SYSTEM BUS
390

FIG. 8

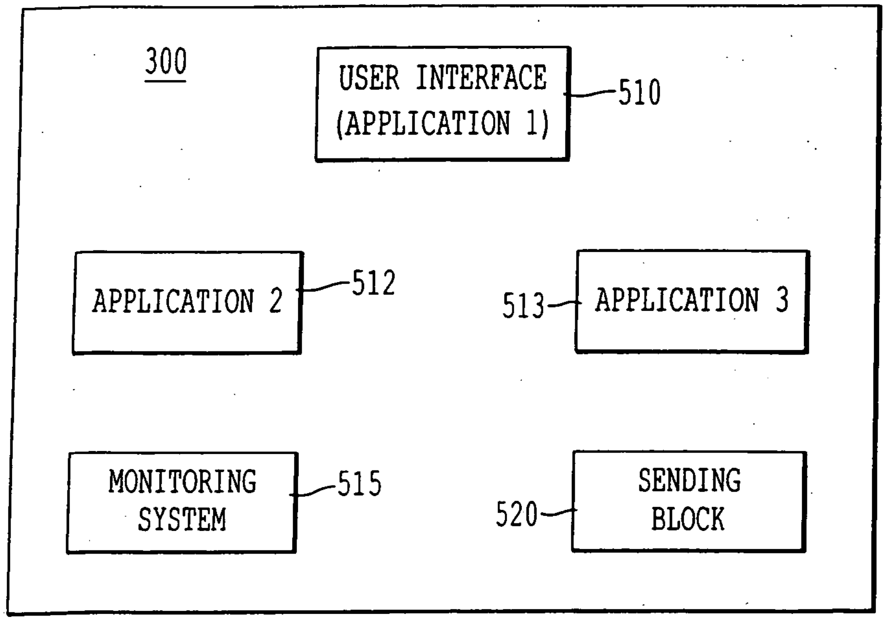

300
USER INTERFACE (APPLICATION 1)
510
APPLICATION 2
512
513
APPLICATION 3
MONITORING SYSTEM
515
520
SENDING BLOCK

FIG. 9

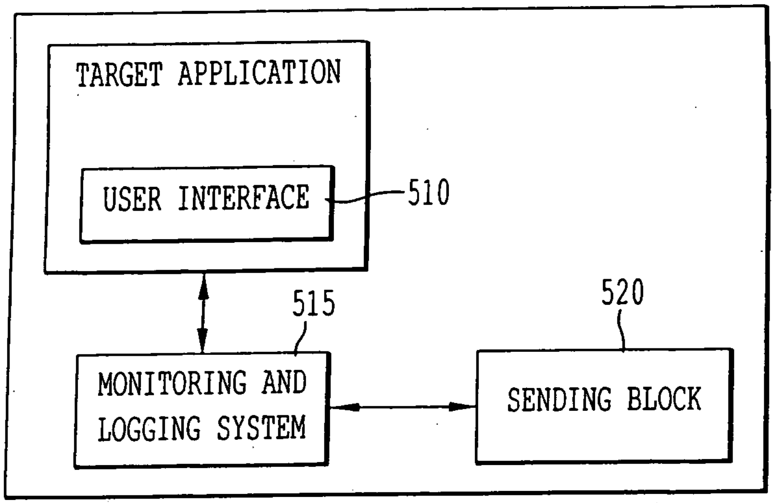

TARGET APPLICATION
USER INTERFACE
510
515
520
MONITORING AND LOGGING SYSTEM
SENDING BLOCK

FIG. 10

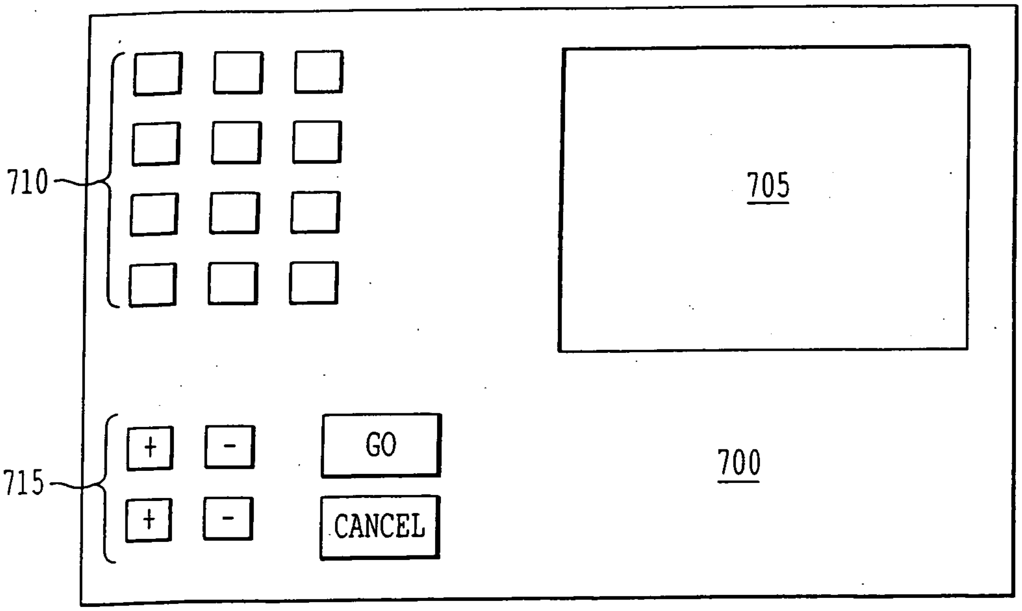

710
705
715
+
-
GO
+
-
CANCEL
700

FIG. 11

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

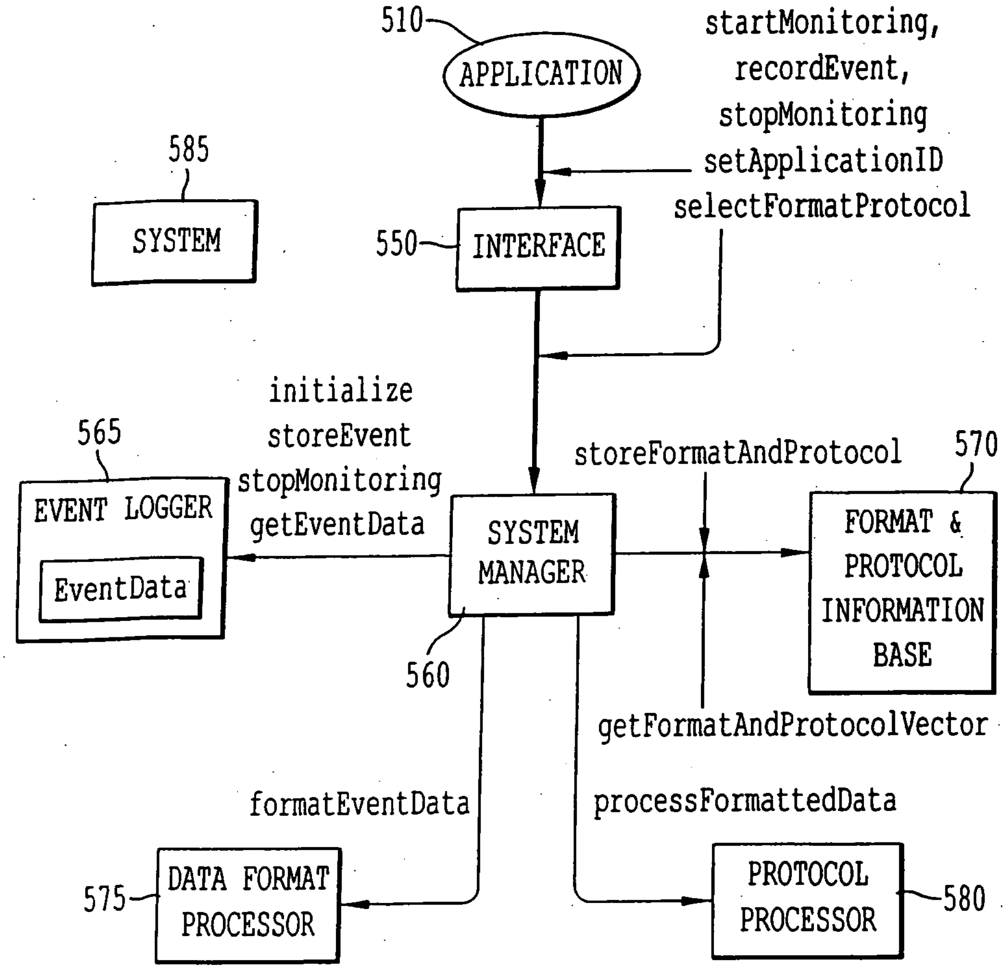


FIG. 12A

RETURN VALUE	FUNCTION NAME	DESCRIPTION
bool	getNextSession	RETURNS FALSE WHEN THERE IS NO MORE SESSION; TRUE OTHERWISE
string	getFileName	RETURNS FILE NAME FOR THE EventData
map<string,string>	getSessionInformation	RETURNS THE MAP. KEYS ARE UserID, Application ID, CumulativeSessionNumber, StartTime, and Duration
map<string, vector<string>>	getSessionEventData	RETURNS THE MAP. KEYS ARE EventName and EventTiming. THE VALUES OF EventTiming VECTOR ARE IN THE UNIT OF 10th OF A SECOND CONVERTED FROM UNSIGNED INTEGER TO STRING

FIG. 12B

RETURN VALUE	FUNCTION NAME	DESCRIPTION
bool	getNextLine	RETURNS ONE LINE OF STRING DATA AS AN OUT PARAMETER STRING. THE FUNCTION RETURNS TRUE IF THERE IS A LINE; FALSE IF NO MORE LINE EXISTS WITH EMPTY STRING
string	getFileNameWithSuffix	RETURNS FILE NAME FOR THE DATA WITH SUFFIX IF APPLICABLE

FIG. 12C

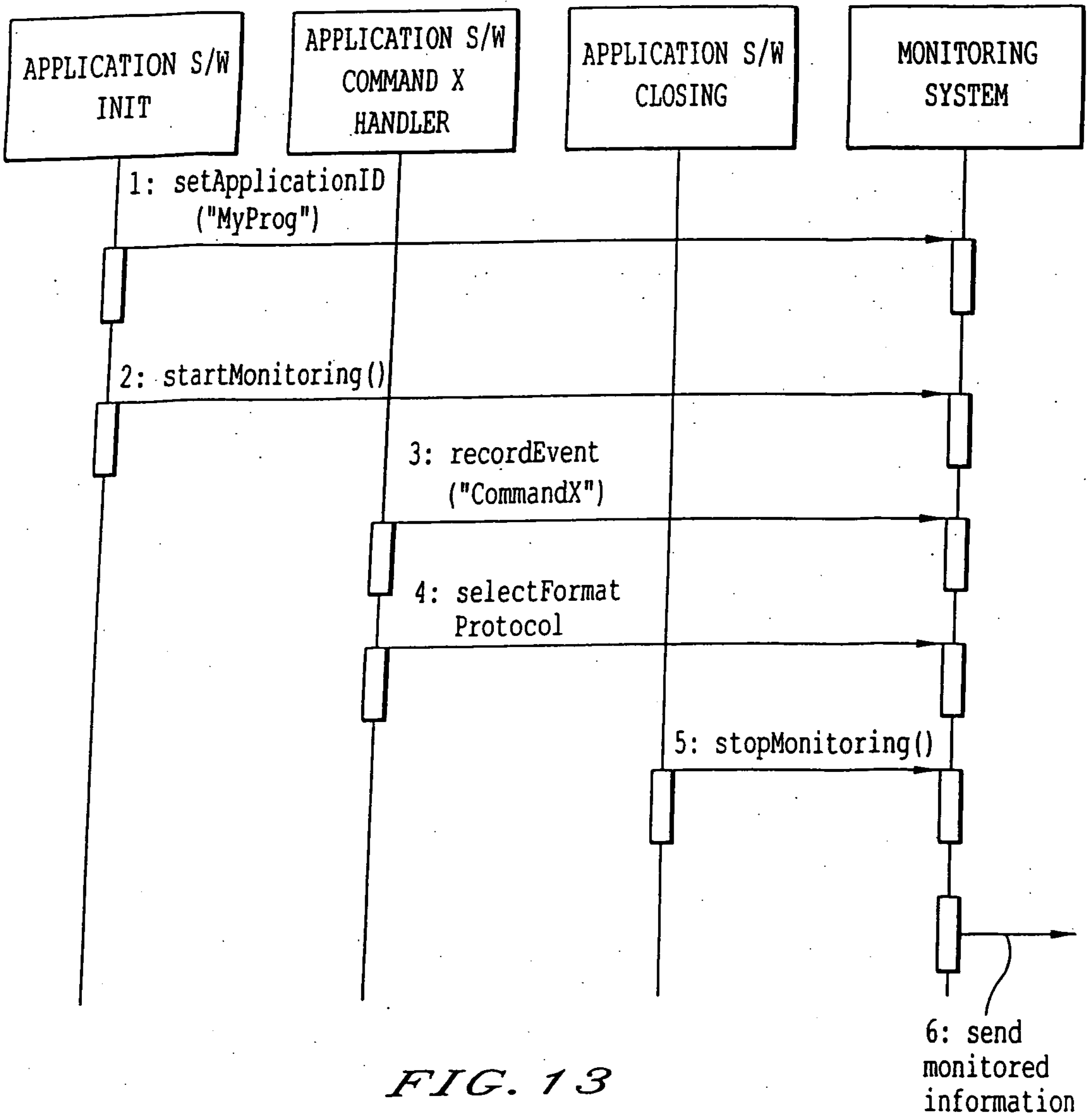
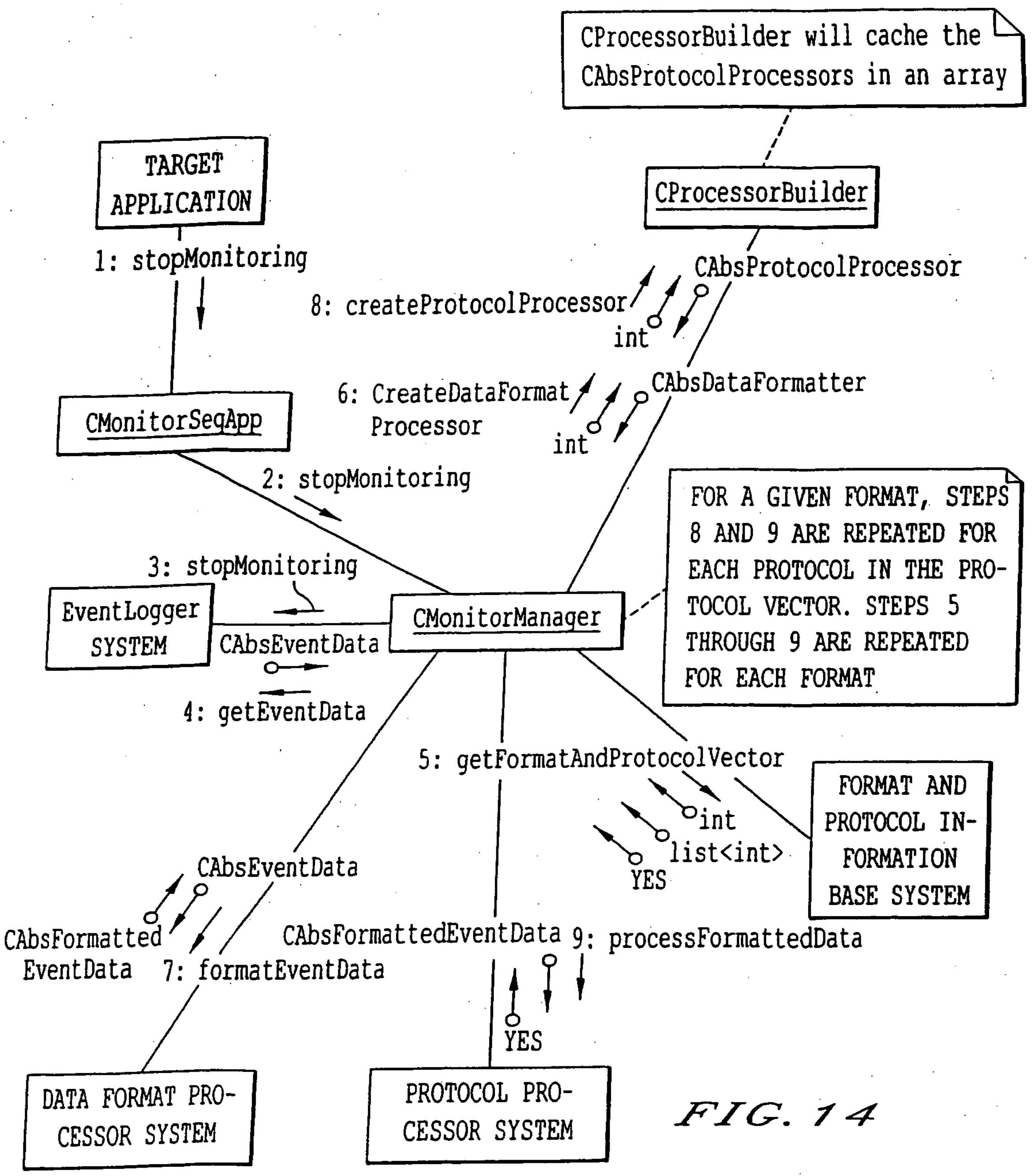

APPLICATION S/W INIT
APPLICATION S/W COMMAND X HANDLER
APPLICATION S/W CLOSING
MONITORING SYSTEM
1: setApplicationID ("MyProg")
2: startMonitoring()
3: recordEvent ("CommandX")
4: selectFormat Protocol
5: stopMonitoring()
6: send monitored information

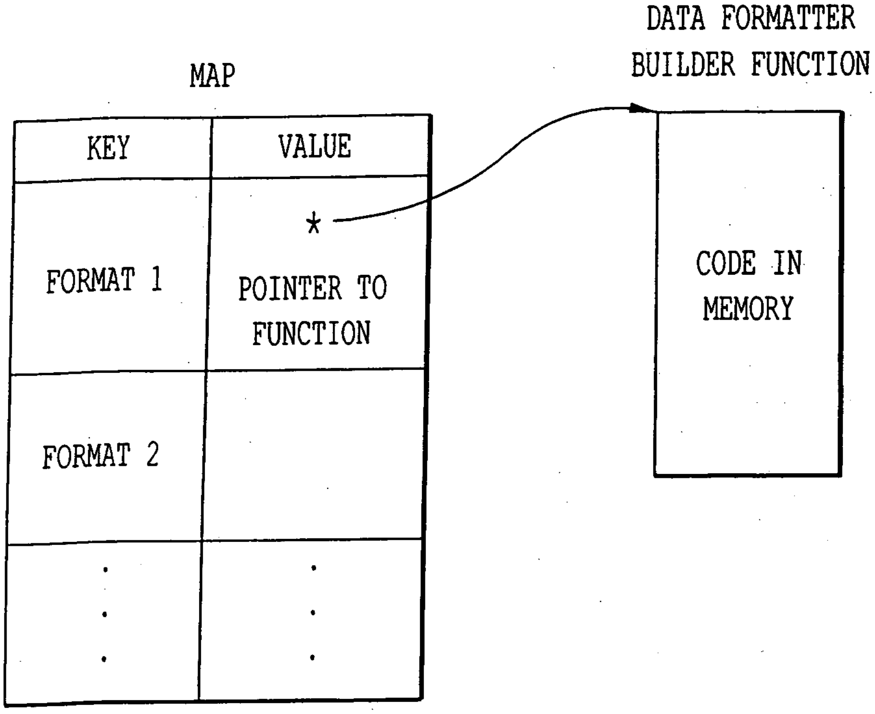
FIG. 13

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS



CProcessorBuilder will cache the CAbsProtocolProcessors in an array
TARGET APPLICATION
CProcessorBuilder
1: stopMonitoring
8: createProtocolProcessor
int
CAbsProtocolProcessor
6: CreateDataFormat Processor
int
CAbsDataFormatter
CMonitorSeqApp
2: stopMonitoring
FOR A GIVEN FORMAT, STEPS 8 AND 9 ARE REPEATED FOR EACH PROTOCOL IN THE PROTOCOL VECTOR. STEPS 5 THROUGH 9 ARE REPEATED FOR EACH FORMAT
3: stopMonitoring
EventLogger SYSTEM
CMonitorManager
CAbsEventData
4: getEventData
5: getFormatAndProtocolVector
int
list<int>
YES
FORMAT AND PROTOCOL INFORMATION BASE SYSTEM
CAbsEventData
CAbsFormatted EventData
7: formatEventData
CAbsFormattedEventData
9: processFormattedData
YES
DATA FORMAT PROCESSOR SYSTEM
PROTOCOL PROCESSOR SYSTEM

FIG. 14



m_DataFormatProcessorMap
(in FIG.18A)

FIG. 15

```
void CMonitorManager::stopMonitoring()
           TRACE ("CMonitorManager::stopMonitoring \n");
// 1.      calls the function stopMonitoring() of
//         CUsageLogger.
           m_UsageLogger.stopMonitoring();
// 2.      calls the function getEvenData()of
//         CUsageLogger. This function returns the usage
//         information, CAbsEventData, to CMonitorManager.
           CAbsEventData * loc_pAbsEventData = m_UsageLogger.getEventData();
// 3.      calls the function getFormatAndProtocolVector()
//         of CFormatProtocol_InformationBase. This  function
//         returns the following to CMonitorManager: an int for
//         the data format, a list<int> for the communication
//         protocols,and a bool to indicate if the return
//         values (format and protocol) are valid.
           int loc_nFormat;
           list<int>loc_ProtocolVector;
           CProcessorBuilder loc_ProcessorBuilder;
           while(m_FormatProtocol_InformationBase.getFormatAndProtocolVector(
           loc_nFormat, loc_ProtocolVector))(
// 4.      calls the function createDataFormatProcessor()
//         of CProcessorBuilder. CMonitorManager passes an
//         int for the data format into this function.  This
//         function returns the data format processor,
//         CAbsDataFormatter, to CMonitorManager.
           CAbsDataFormatter * loc_pAbsDataFormatter =
           loc_ProcessorBuilder.createDataFormatProcessor(loc_nFormat);
```

FIG. 16A

```
// 5.   calls the function formatEventData() of
//      CAbsDataFormatter. CMonitorManager passes the
//      usage information, CAbsEventData, into this
//      function. This function returns the formatted
//      usage information, CAbsFormattedEventData, to
//      CMonitorManager.
        CAbsFormattedEventData * loc_pAbsFormattedEventData =
        loc_pAbsDataFormatter->formatEventData(loc_pAbsEventData);
// 6.   calls the function createProtocolProcessor() of
//      CProcessorBuilder. CMonitorManager passes an int
//      for the communication protocol into this function.
//      The int is the first int from the protocol vector,
//      list<int>. This function returns the protocol
//      processor, CAbsProtocolProcessor, to CMonitorManager.
        for(list<int>::iterator loc_ProtocolVectorIterator =
        loc_ProtocolVector.begin(); loc_ProtocolVectorIterator NE
        loc_ProtocolVector.end(); loc_ProtocolVectorIterator ++)(
        CAbsProtocolProcessor * loc_pAbsProtocolProcessor =
        loc_ProcessorBuilder.createProtocolProcessor(
        * loc_ProtocolVectorIterator);
// 7.   calls the function processFormattedData() of
//      CAbsProtocolProcessor. CMonitorManager passes the
//      formatted usage information, CAbsFormattedEventData,
//      into this function. This function returns a bool to
//      CMonitorManager to indicate if the usage information
//      was communicated using the protocol.
        loc_pAbsProtocolProcessor->processFormattedData(
        loc_pAbsFormattedEventData);
                       )
// 8.   steps 6 and 7 are repeated for each protocol,
//      int, in the protocol vector, list<int>.
        )
// 9.   steps 3 through 8 are repeated for each format
//      until the function getFormatAndProtocolVector()
//      returns NO to CMonitorManager.
)
```

FIG. 16B

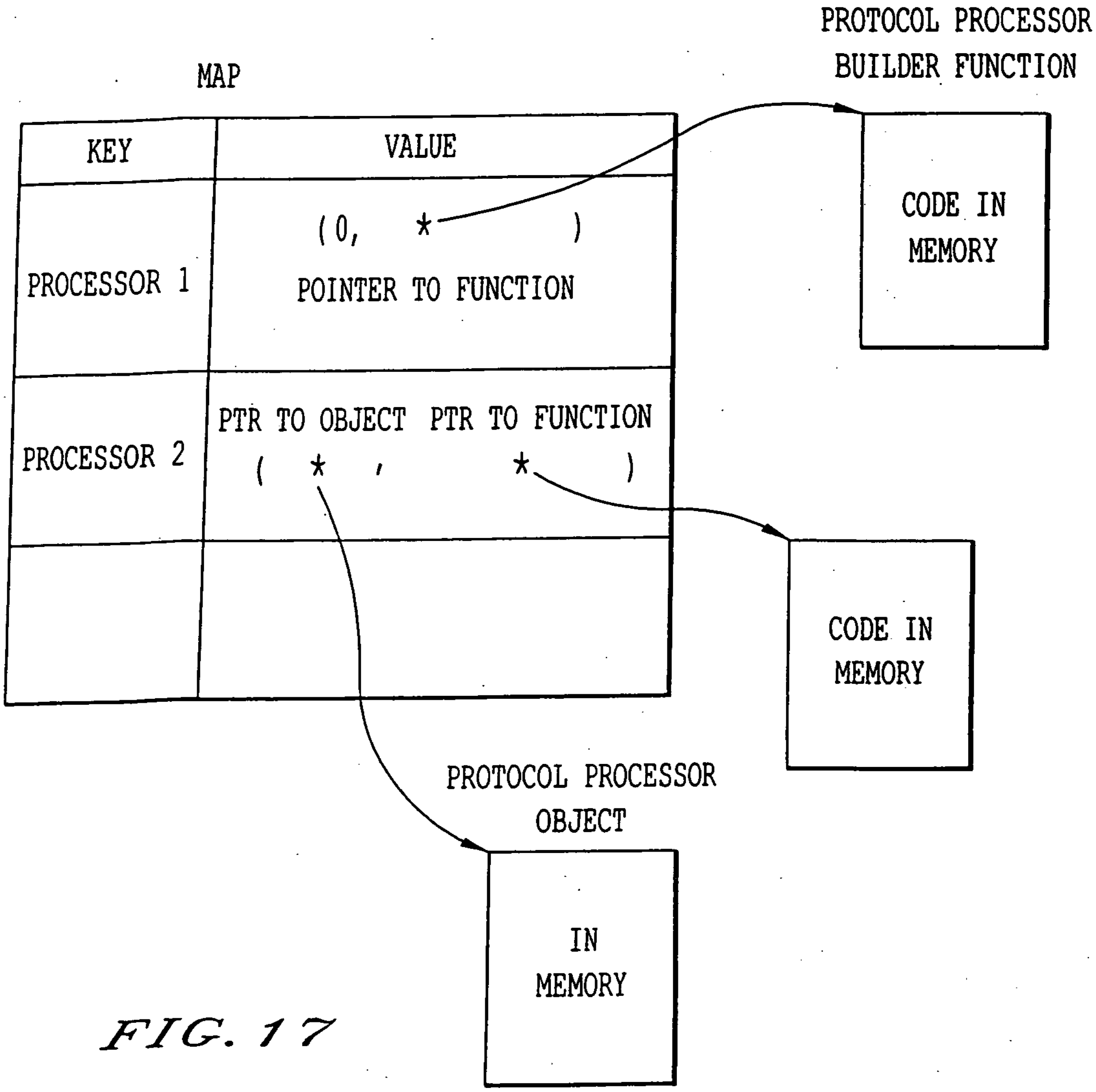
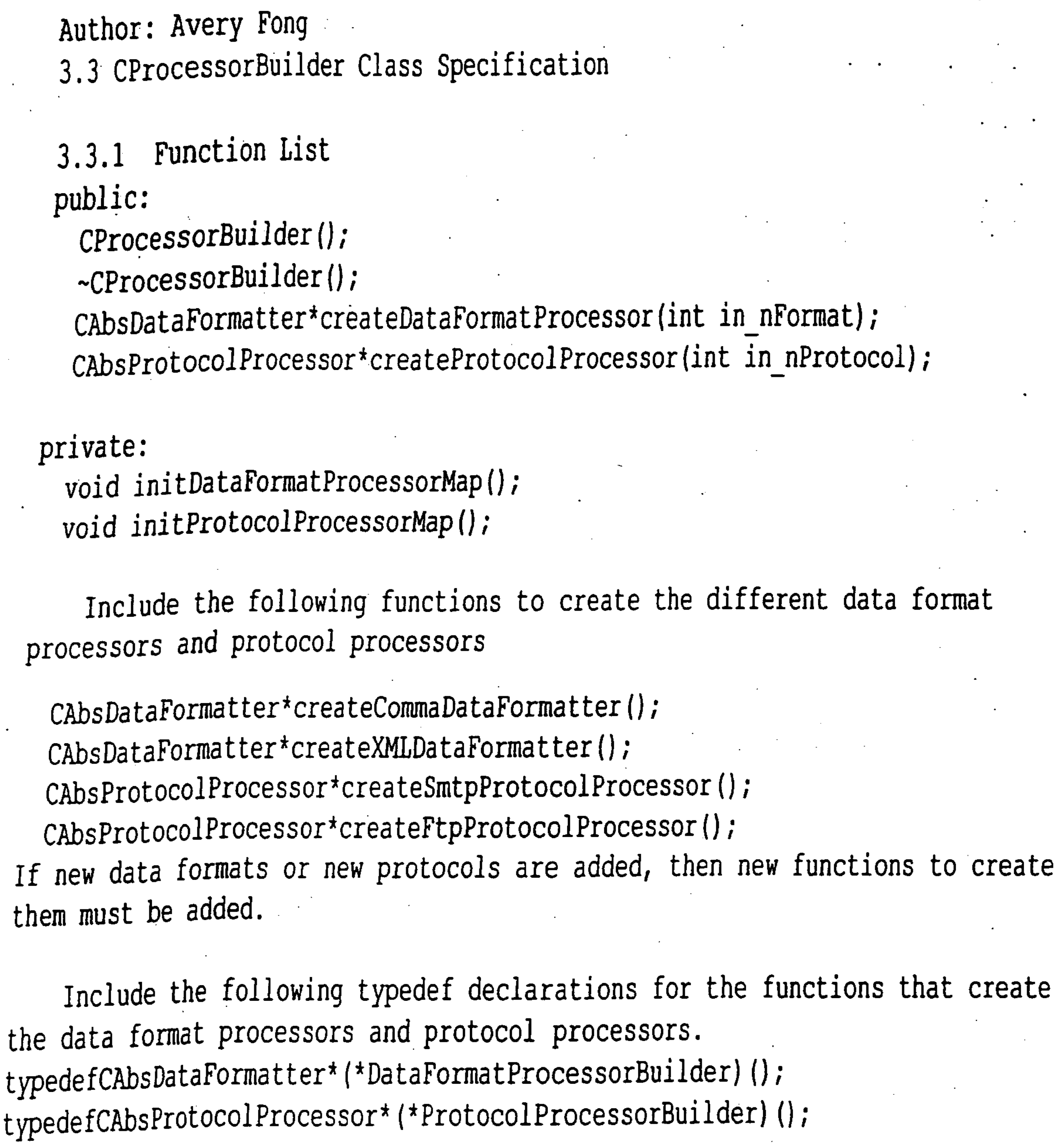

PROTOCOL PROCESSOR
BUILDER FUNCTION
MAP
KEY
VALUE
PROCESSOR 1
(0, *)
POINTER TO FUNCTION
CODE IN
MEMORY
PROCESSOR 2
PTR TO OBJECT PTR TO FUNCTION
(* , *)
CODE IN
MEMORY
PROTOCOL PROCESSOR
OBJECT
IN
MEMORY

FIG. 17



Author: Avery Fong

3.3 CProcessorBuilder Class Specification

3.3.1 Function List

```
public:
  CProcessorBuilder();
  ~CProcessorBuilder();
  CAbsDataFormatter*createDataFormatProcessor(int in_nFormat);
  CAbsProtocolProcessor*createProtocolProcessor(int in_nProtocol);

private:
  void initDataFormatProcessorMap();
  void initProtocolProcessorMap();
```

Include the following functions to create the different data format processors and protocol processors

```
CAbsDataFormatter*createCommaDataFormatter();
CAbsDataFormatter*createXMLDataFormatter();
CAbsProtocolProcessor*createSmtpProtocolProcessor();
CAbsProtocolProcessor*createFtpProtocolProcessor();
```

If new data formats or new protocols are added, then new functions to create them must be added.

Include the following typedef declarations for the functions that create the data format processors and protocol processors.

```
typedefCAbsDataFormatter*(*DataFormatProcessorBuilder)();
typedefCAbsProtocolProcessor*(*ProtocolProcessorBuilder)();
```

FIG. 18A

3.3.2 Class Attributes

Type	Attribute Name	Description
CAbsDataFormatter*	m_pDataFormatter	This attribute member points to the data format processor object It is initialize to 0 in the constructor and the data format processor object is created by the function createDataFormatProcessor(). This function may be called multiple times so that it must delete the previous data format processor object pointed to by this attribute member before creating a new one. The destructor will delete the last data format processor object pointed to by this attribute member.
map<int, DataFormatProcessor Builder>	m_ProtocolProcessorMap	This attribute member is a map of pointers to functions that create the data format processor. The key to this map is an int for the data format type. The value is a pointer to a function that creates the data format processor corresponding to the key. The pointers to the functions in the map are initialized in the function initDataFormatProcessorMap().

Continued to Fig. 18C

FIG. 18B

Continued from Fig.18B

map<int, pair<CAbsProtocol Processor*,Protocol ProcessorBuilder>>	m_ProtocolProcessorMap	This attribute member is a map of pointers to protocol processor objects and pointers to functions that create them. The key to this map is an int for the protocol processor type. The value is a pair consisting of a pointer to the protocol processor object and a pointer to a function that creates the protocol processor object. All the pointers to the protocol processor object are initialized to 0 and its corresponding functions are initialized by the function initProtocolProcessorMap(). The protocol processor objects are created by the function createProtocolProcessor(). The destructor will delete all the protocol processor objects pointed to by the map.

FIG. 18C

3.3.3 Function Definitions

```
////////////////////////////////////////////////////////////////////////////////
// Function:         CProcessorBuilder
// Description:      Constructor
// Preconditions:    None.
// Postconditions:   None.
// Algorithm:        1.    calls the private function
//                   initDataFormatProcessorMap().
//                   2.  calls the private function
//                   initProtocolProcessorMap().
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Function:         ~CProcessorBuilder
// Description:      Destructor
// Preconditions:    None.
// Postconditions:   None.
// Algorithm:        1.  delete the object pointed to by m_pDataFormatter.
//                   2.  iterate through the map, m_ProtocolProcessorMap.
//                   For each entry in the map, get the protocol
//                   processor object pointed to by the pair and delete
//                   the object.
////////////////////////////////////////////////////////////////////////////////
```

FIG. 18D

```
//////////////////////////////////////////////////////////////////////////////////
// Function:        createDataFormatProcessor
// Description:     This function creates a data format processor
//                  object.  The data format processor object created
//                  corresponds to the data format type in_nFormat.
// Preconditions:   The data format type must be valid.
// Postconditions:  The pointer to the data format processor object,
//                  m_pDataFormatter, cannot be 0.
// Algortihm:       1.  if m_pDataFormatter currently points to a data
//                  format processor object, then delete the object.
//                  2.  creates a new data format processor object by
//                  calling the function in the map,
//                  m_DataFormatProcessorMap, that corresponds to the
//                  data format type, in_nFormat, and assign it to
//                  m_pDataFormatter.
//                  3.  returns m_pDataFormatter.
//////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////////
// Function:        createProtocolProcessor
// Description:     This function creates a protocol processor object.
//                  The protocol processor object created corresponds
//                  to the protocol type in_nProtocol.
// Preconditions:   The protocol type must be valid.
// Postconditions:  The pointer to the created protocol processor object
//                  cannot be 0.
// Algortihm:       1.  for the protocol type, in_nProtocol,  get the
//                  pair from the map that contains the pointer to
//                  protocol processor object and its corresponding
//                  pointer to the function that creates it.
//                  2.  if the pointer to the protocol processor object
//                  is 0, then use its corresponding function to create
//                  it and assign it to the pointer in the map.  Return
//                  the pointer to the protocol processor object.
//                  3.  if the pointer points to a protocol processor
//                  object, then return this pointer.
//////////////////////////////////////////////////////////////////////////////////
```

FIG. 18E

```
/////////////////////////////////////////////////////////////////////////////
// Private
// Function:        initDataFormatProcessorMap
// Description:     This function initializes all the function pointers
//                  in the map m_DataFormatProcessorMap.  If new data
//                  formats are added, then this function must be
//                  modified.
// Preconditions:   None.
// Postconditions:  None.
// Algorithm:       1.  add entries to the map, m_DataFormatProcessorMap,
//                  for each data format type.  The key will be the
//                  data format type and the value will be the pointer
//                  to the corresponding function that creates the
//                  data format processor.
//                  2.  for data format type 1, the function pointer
//                  points to createCommaDataFormatter ().
//                  3.  for data format type 2, the function pointer
//                  points to createXMLDataFormatter ().
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Private
// Function:        initProtocolProcessorMap
// Description:     This function initializes all the pairs of pointers
//                  in the map m_ProtocolProcessorMap.  If new protocols
//                  are added, then this function must be modified.
// Preconditions:   None.
// Postconditions:  None.
// Algorithm:       1.  add entries to the map, m_ProtocolProcessorMap,
//                  for each protocol type.  The key will be the
//                  protocol type and the value will be a pointer to
//                  the protocol processor object and a pointer
//                  to the corresponding function that creates the
//                  protocol processor.  All ponters to the protocol
//                  processor objects will be set to 0.
//                  2.  for protocol type 1, the function pointer
//                  points to createSmtpProtocolProcessor ().
//                  3.  for protocol type 2, the function pointer
//                  points to createFtpProtocolProcessor ().
/////////////////////////////////////////////////////////////////////////////
```

FIG. 18F

```
////////////////////////////////////////////////////////////////////////////////
// Function:        createCommaDataFormatter
// Description:     This function creates and returns a comma data
//                  formatter object.
// Preconditions:   None.
// Postconditions:  The pointer to the created comma data formatter
//                  object cannot be 0.
// Algorithm:       1.  creates and returns an object of the class
//                  CCommaDataFormatter.
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Function:        createXMLDataFormatter
// Description:     This function creates and returns a XML data
//                  formatter object.
// Preconditions:   None.
// Postconditions:  The pointer to the created XML data formatter
//                  object cannot be 0.
// Algorithm:       1.  creates and returns an object of the class
//                  CXMLDataFormatter.
////////////////////////////////////////////////////////////////////////////////
```

FIG. 18G

```
////////////////////////////////////////////////////////////////////////////
// Function:        createSmtProtocolProcessor
// Description:     This function creates and returns an SMTP protocol
//                  processor object.
// Preconditions:   None.
// Postconditions:  The pointer to the created smtp protocol processor
//                  object cannot be 0.
// Algorithm:       1.  creates and return an object of the class
//                  CSmtpProtocolProcessor
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Function:        createFtpProtocolProcessor
// Description:     This function creates and returns an FTP protocol
//                  processor object.
// Preconditions:   None.
// Postconditions:  The pointer to the created ftp protocol processor
//                  object cannot be 0.
// Algorithm:       1.  creates and returns an object of the class
//                  CFtpProtocolProcessor.
////////////////////////////////////////////////////////////////////////////
```

FIG. 18H

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

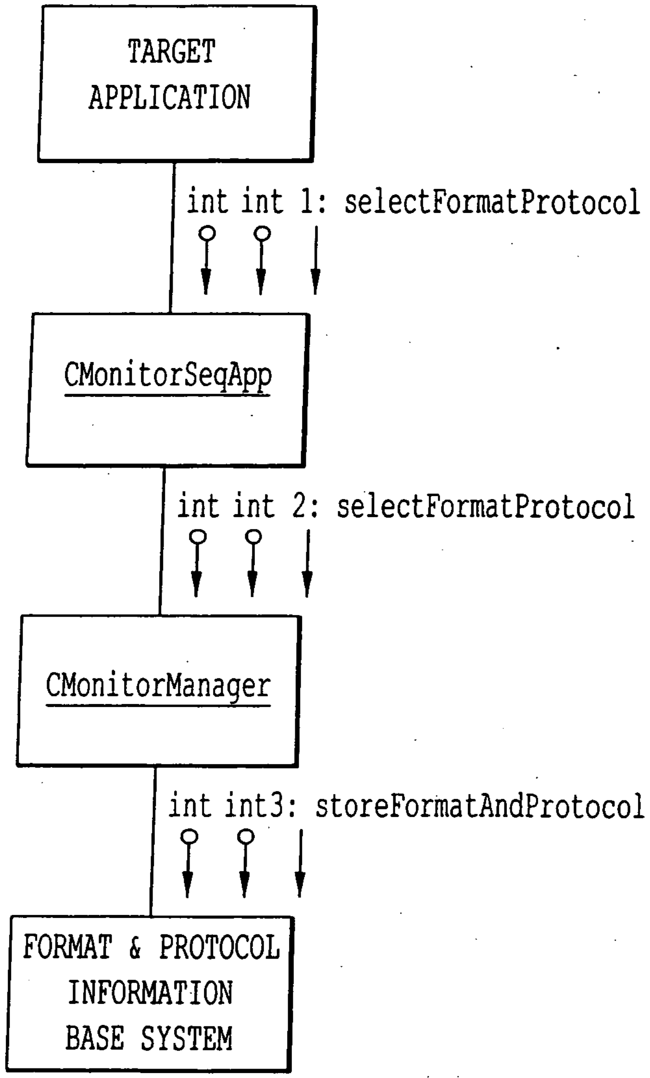
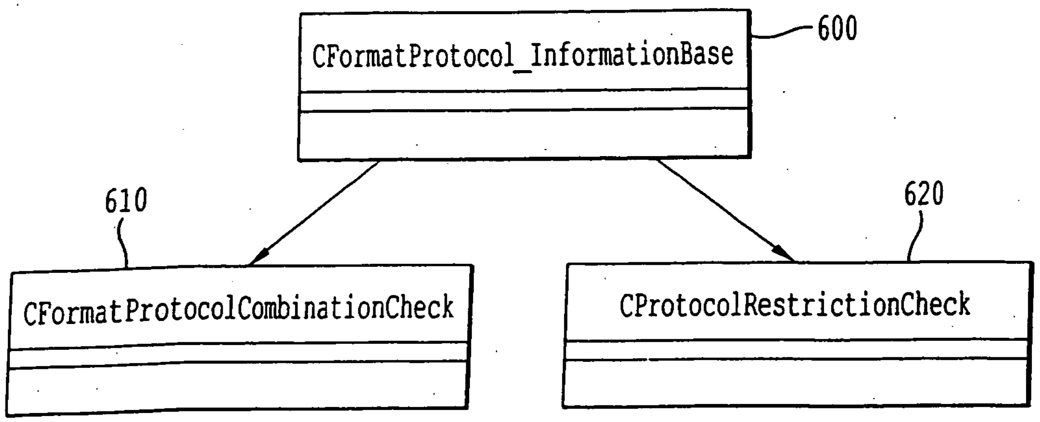

TARGET APPLICATION
int int 1: selectFormatProtocol
CMonitorSeqApp
int int 2: selectFormatProtocol
CMonitorManager
int int3: storeFormatAndProtocol
FORMAT & PROTOCOL INFORMATION BASE SYSTEM

FIG. 19

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS


CFormatProtocol_InformationBase
600
610
620
CFormatProtocolCombinationCheck
CProtocolRestrictionCheck

FORMAT AND PROTOCOL INFORMATION BASE PACKAGE CLASS STRUCTURE

FIG. 20

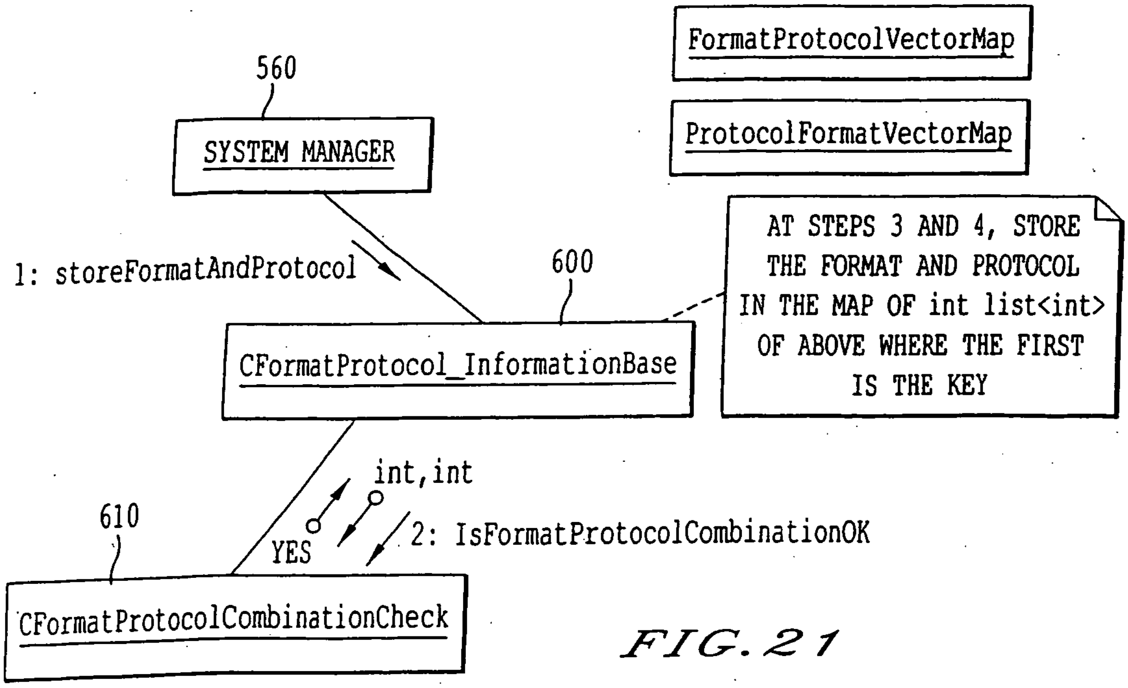

FormatProtocolVectorMap
ProtocolFormatVectorMap
560
SYSTEM MANAGER
1: storeFormatAndProtocol
600
CFormatProtocol_InformationBase
AT STEPS 3 AND 4, STORE THE FORMAT AND PROTOCOL IN THE MAP OF int list<int> OF ABOVE WHERE THE FIRST IS THE KEY
int,int
610
YES
2: IsFormatProtocolCombinationOK
CFormatProtocolCombinationCheck

FIG. 21

OBLON, SPIVAK, ET AL
Docket #: 5244-0121-2
Inventor: Tetsuro MOTOYAMA
Serial No: 09/453,934
Reply to Office Action dated: 09-09-2005
REPLACEMENT SHEETS

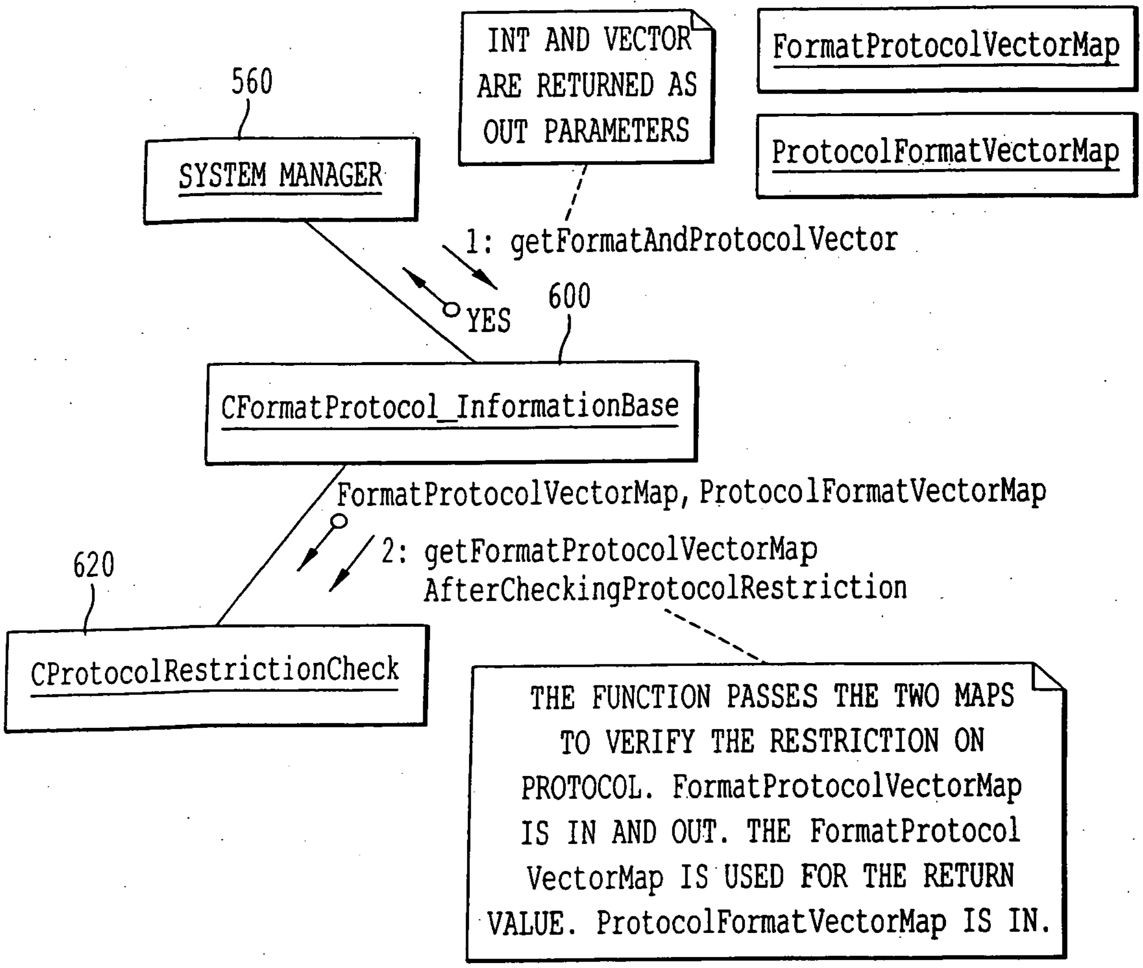

INT AND VECTOR ARE RETURNED AS OUT PARAMETERS
FormatProtocolVectorMap
ProtocolFormatVectorMap
560
SYSTEM MANAGER
1: getFormatAndProtocolVector
YES
600
CFormatProtocol_InformationBase
FormatProtocolVectorMap, ProtocolFormatVectorMap
2: getFormatProtocolVectorMap AfterCheckingProtocolRestriction
620
CProtocolRestrictionCheck
THE FUNCTION PASSES THE TWO MAPS TO VERIFY THE RESTRICTION ON PROTOCOL. FormatProtocolVectorMap IS IN AND OUT. THE FormatProtocol VectorMap IS USED FOR THE RETURN VALUE. ProtocolFormatVectorMap IS IN.

FIG. 22

CFormatProtocol_InformationBase Class Specification

Author: Tetsuro Motoyama

5.2 CFormatProtocol_InformationBase Class Specification

5.2.1 Function List

```
public:
   CFormatProtocol InformationBase();
   ~CFormatProtocol_InformationBase();
   void storeFormatAndProtocol(int in_nFormat, int in_nProtocol);
   bool getFormatAndProtocolVector(int & out_nFormat, list<int> & out_ProtocolVector);
private:
   void setDefaultFormatAndProtocol();
```

5.2.2 Class Attributes

Type	Attribute Name	Description
map<int, list<int>>	m_FormatProtocolVectorMap	The key is a format value, and the list is the list of protocol values associated to the key. Because subscripting □ is not needed in this implementation, list is used for the vector implementation. This map is used to return the necessary information for getFormatAndProtocol Vector function Note: >>is>space> to distinguish from'>>' that is used by iostream.
map<int, list<int>>	m_ProtocolFormatVectorMap	The key is a protocol value, and the list is the list of format values associated to the key. Because subscripting □ is not needed in this implementation, list is used for the vector implementation. This map is used to modify the map above if the protocol can take only one format.

Continued to FIG. 23B

FIG. 23A

Continued From FIG. 23A

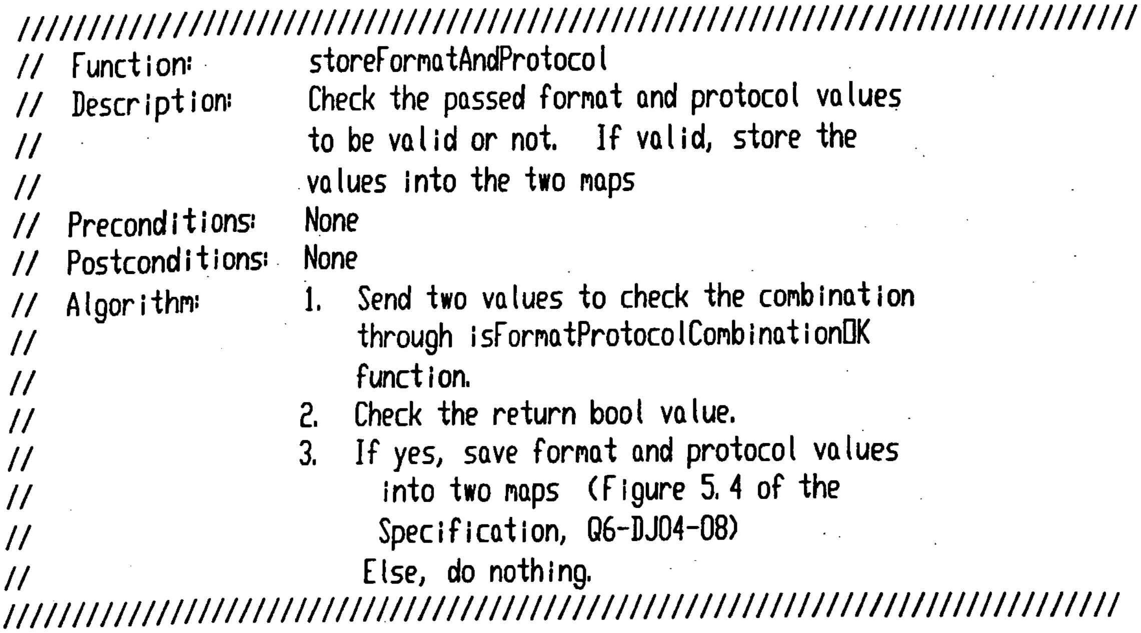
bool	m_bFirstGetCall	This flag is used to call the function in CProtocolRestrictionCheck. The constructor set this to be true. The function, getFormatAndProtocol Vector, sets it to be false
map(int, list(int))::iterator	m_FormatProtocolVectorMapIterator	interator used to iterate the map.
CFormatProtocol CombinationCheck	m_FormatProtocol CombinationCheck	This object is to check the combination of format and protocol
CProtocolRestriction Check	m_ProtocolRestriction Check	This object is to check the protocol restriction. Currently, the only restriction is if protocol can have only one format support.

5.2.3 Function Definitions

```
//////////////////////////////////////////////////////////////////////////////////
//  Function:        CFormatProtocol_InformationBase
//  Description:     Constructor
//  Preconditions:   None
//  Postconditions:  None
//  Algorithm:       Set m_bFirstGetCall to true
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
//  Function:        ~CFormatProtocol_InformationBase
//  Description:     Destructor
//  Preconditions:   None
//  Postconditions:  None
//  Algorithm:       Default
//////////////////////////////////////////////////////////////////////////////////
```

FIG. 23B



```
//////////////////////////////////////////////////////////////////////////////
// Function:        storeFormatAndProtocol
// Description:     Check the passed format and protocol values
//                  to be valid or not.  If valid, store the
//                  values into the two maps
// Preconditions:   None
// Postconditions:  None
// Algorithm:       1.  Send two values to check the combination
//                      through isFormatProtocolCombinationOK
//                      function.
//                  2.  Check the return bool value.
//                  3.  If yes, save format and protocol values
//                        into two maps (Figure 5.4 of the
//                        Specification, Q6-DJ04-08)
//                      Else, do nothing.
//////////////////////////////////////////////////////////////////////////////
```

FIG. 23C

```
/////////////////////////////////////////////////////////////////////////
// Function:        getFormatAndProtocolVector
// Description:     The function returns a format and a list
//                  of protocol values associated with the
//                  format through two parameters.  The function
//                  returns true if a format and list are
//                  returned, false otherwise.
// Preconditions:   None
// Postconditions:  The format value is within the range.
//                  The list is not empty and contains the values
//                  within the range.
// Algorithm:       1.  If m_bFirstGetCall (Figure 5.5 of the
//                        Specification Q6-DJ04-08)
//                     1.1  call the function to check the protocol
//                          restriction.
//                     1.2  check if m_FormatProtocolVectorMap is
//                          empty.  If empty, set it to default
//                          values of format and protocol by calling
//                          setDefaultFormatAndProtocol function.
//                     1.3  set the iterator to begin ().
//                     1.4  set m_bFirestGetCall to be false
//                  2.  If iterator is end, return false.
//                     else  (Figure 5.6 of the Specification
//                              Q6-DJ04-08)
//                     get format and list to return and set
//                     return parameters.
//                     increment iterator.
//                     Return true.
/////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////
// Function:        setDefaultFormatAndProtocol
// Description:       The functions sets the default values for format and protocol
// Preconditions:     The m_FormatProtocolVectorMap is empty.        in the map
// Postconditions:    The map contains one default format and a
//                  protocol list with one default protocol.
// Algorithm:        Set the map with the default values.
/////////////////////////////////////////////////////////////////////////
```

FIG. 23D

CFormatProtocolCombinationCheck Class Specification

Author: Tetsuro Motoyama

5.3 CFormatProtocolCombinationCheck Class Specification

5.3.1 Function List

```
public:
   CFormatProtocolCombinationCheck();
   ~CFormatProtocol CombinationCheck()
   bool isFormatProtocolCombination OK(const int in_nFormat, const int in_nProtocol);

private:
   void initMatrix();
```

5.3.2 Class Attributes

Type	Attribute Name	Description
map<int, set<int>>	m_CombinationMatrix	Key is the format. The set contains the protocols that are valid for the particular format

5.3. Function Definitions

```
//////////////////////////////////////////////////////////////////////////////////
// Function:         CFormatProtocolCombinationCheck
// Description:      Constructor
// Preconditions:    None
// Postconditions:   None
// Algorithm:        call initMatrix
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
// Function:         ~CFormatProtocolCombinationCheck
// Description:      Destructor
// Preconditions:    None
// Postconditions:   None
// Algorithm:        Default
//////////////////////////////////////////////////////////////////////////////////
```

FIG. 24A

```
//////////////////////////////////////////////////////////////////////////////
// Function:          isFormat ProtocolCombinationOK
// Description:       Check the passed format and protocol values
//                    to be valid or not.  If valid, return yes
//                    no otherwise
// Preconditions:     None
// Postconditions:    None
// Algorithm:         1.  Use find function of the Matrix for
//                         in_nFormat
//                    2.  If returned iterator is end, return No
//                    3.  get the set value for the key format
//                    4.  Use the find function of the set for
//                        in_nProtocol
//                    5.  if returned iterator is end, return no
//                    6.  return yes
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
// Private Function:  initMatrix
// Description:       This function initializes m_CombinationMatrix.
//                    If new formats or protocols are added, this
//                    function must be modified.
// Preconditions:     None
// Postconditions:    None
// Algorithm:         1.  Create the local set<int>
//                    2.  for each format
//                         2.1 fill in the local set
//                         with the protocol numbers
//                         that are valid for the format,
//                         using insert function
//                         2.2 m_CombinationMatrix [format]
//                               = local set
//                         2.3 clear local set
//////////////////////////////////////////////////////////////////////////////
```

FIG. 24B

CProtocolRestrictionCheck Class Specification

Author: Tetsuro Motoyama

5.4 CFormatProtocolRestrictionCheck Class Specification

5.4.1 Function List

```
public:
   CFormatProtocolRestrictionCheck();
   ~CFormatProtocolRestrictionCheck()
   void getFormatProtocolVectorMapAfterCheckingProtocolRestriction
     (map<int, list<int>> & inOut_Map, const map<int, list<int, list<int>> & in_Map);

private:
   void initOneFormatRestriction();
   void oneFormatRestriction()
    (map<int, list<int>> & inOut_Map, const map<int, list<int>> & in_Map);
```

5.4.2 Class Attributes

Type	Attribute Name	Description
vector<bool>	m_bOneFormatRestriction	Array size should be protocol size+1. The position corresponds to the protocol.

5.4.3. Function Definitions

```
/////////////////////////////////////////////////////////////////////////
// Function:         CProtocolRestrictionCheck
// Description:      Constructor
// Preconditions:    None
// Postconditions:   None
// Algorithm:        call initOneFormatRestriction
/////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////
// Function:         ~CFormatProtocolRestrictionCheck
// Description:      Destructor
// Preconditions:    None
// Postconditions:   None
// Algorithm:        Default
/////////////////////////////////////////////////////////////////////////
```

FIG. 25A

```
//////////////////////////////////////////////////////////////////////////////
// Function:          getFormatProtocolVectorMapAfterCheckingProtocolRestriction
// Description:       Check the restriction on the protocol.
//                    Currently, there is only one possible restriction
//                    defined in the requirements.  If there are more
//                    restrictions, more private functions should be
//                    added and called.
// Preconditions:     None
// Postconditions:    None
// Algorithm:         1.  Call oneFormatRestriction function
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
// Private Function:  initOneFormatRestriction
// Description:       This function initialize the attribute
//                    m_bOneFormatRestriction.  If more portocols are
//                    added, this initialization must be modified.
// Preconditions:     None
// Postconditions:    None
// Algorithm:         1.  use assign(size+1, false) to initialzie the
//                    vector to false.
//                    2.  set the entries of true.
//                    Note:  for class debug version, use
//                        ifdef and
//                        bool & pos1 = m_bOne FormatRestriction [1];
//                        bool & pos2 = m_bOneFormatRestriction [2];
//                        and so on to be able to see and to
//                        change the value.
//////////////////////////////////////////////////////////////////////////////
```

FIG. 25B

```
//////////////////////////////////////////////////////////////////////////////
// Private Function:   oneFormatRestriction
// Description:        This function receives two maps and if the one
//                     restriction is true for given protocol, the
//                     content of inOut_Map (m_FormatProtocolVectorMap)
//                     is adjusted accordingly.
// Preconditions:      None
// Postconditions:     None
// Algorithm:          Iterate over the in_Map (m_ProtocolFormatVectorMap)
//                     1.  get the key (pkey)
//                     2.  If m_bOneFormatRestriction[pkey]
//                         2.1  get the value list of in_Map for the key
//                         2.2  local int lastFormat = back (),
//                         2.3  iterate over the list
//                                if *iterator NE lastFormat
//                                 iterate over inOut_Map[*iterator] list
//                                   if the value EQ pkey
//                                      erase the entry from the list
//                     3.  Iterate over inOut_Map
//                           if the value list is empty,
//                              erase the entry from inOut_Map
//----------------------------------------------------------------------------
```

FIG. 25C

```
//  Example:                               0 1 2 3 4
//      m_bOneFormatRestriction = [0,0,1,0,1]  (four protocols)
//                                         0: false,  1: true
//      inOut_Map  (m_Format ProtocolVectorMap)
//          =(1, <1,2,3,4>                              --> <1, 2 ,3>
//             2, <2,1,3,4>                             --> <1, 3>
//             3, <3,4,1,2>                             --> <3, 4, 1>
//             4, <2,4>)                                --> <>
//         in_Map  (m_ProtocolFormatVectorMap)
//          =(1, <1, 3, 2>
//             2, <4, 3, 2, 1>
//             3, <1, 3, 2>
//             4, <4, 2, 1, 3>)
//     pkey = 1  m_bOneFormatRestriction[1] = 0
//     pkey = 2  m_bOneFormatRestriction[2] = 1
//      value list = <4, 3, 2, 1>  (2.1)
//      lastFormat = 1        (2.2)
//     4  ! = 1
//          inOut_Map[4]  = <2, 4>
//          erase value 2     <4>
//     3  ! = 1
//          inOut_Map[3]  = <3, 4, 1, 2>
//          erase value 2   <3, 4, 1>
//     2  ! = 1
//          inOut_Map[2]  = <2, 1, 3, 4>
//          erase value 2     <1, 3, 4>
//     1 == 1
//     pkey = 3 m_bOneFormatResriction[3] = 0
```

FIG. 25D

```
//      pkey = 4  m_bOneFormatRestriction[4] = 1
//        value list = <4, 2, 1, 3>
//        lastFormat = 3
//        4 ! = 3
//          inOut_Map[4]  = <4>
//          erase value  4   <>
//        2 ! = 3
//          inOut_Map[2]  = <1, 3, 4>
//          erase value 4   <1, 3>
//        1 ! = 3
//          inOut_Map[1]  = <1, 2, 3, 4>
//          erase value 4    <1, 2, 3>
//        3 == 3
//      Iterate over inOut_Map
//            if *inOut_Map_iterator.empty() then erase
//
//      inOut_Map
//         = ( 1, <1, 2, 3>
//              2, <1, 3>
//               3, <3, 4, 1>)
////////////////////////////////////////////////////////////
```

FIG. 25E